Lecture 18 - von Neumann Arch cont'd & ISA design

Instruction Format, ISA

- Every instruction has an **opcode** to identify it and **fields** that specify operands and results. It also might have flags to differentiate it from close variants of instructions (e.g. read a character vs. read a string)
- What are some instructions? (Remember we built a very tiny calculator before…)
- We can choose two types of instruction encodings:
    - Fixed length: easier to design, instruction decoding logic much simpler
    - Variable-length: more efficient use of memory, more flexible instruction set, but more complicated hardware (and software)
- The set of instructions supported by the hardware, how they specify operands, their formats etc. is determined by the **Instruction Set Architecture (ISA)**.
- The ISA is the *interface* to the hardware. Think of it as being the minimum amount of detail you need to program the machine.
- Well known ISAs include ARM, x86, x86_64 (x64), OpenSPARC, MIPS, etc.

Addressing Modes

- Conceptually, instructions are simple. But we've really only considered the operators. What about the operands? Where do they come from?
- Usually, registers, memory, or in the instruction itself! We need ways to specify *which* register and *which* memory location
- For this, we use **addressing modes**
- When an operand is part of the actual instruction, we have **immediate operands**
- For example ADDI R0, 1 Might be encoded as 0000 000 0001
- If the data is in a register, we need to be able to specify which, so we need bits for that in the instruction encoding
- What about memory? We could just put the memory address in the instruction. This is **absolute addressing**. The issue here is that our instructions must be big enough to hold the address and the opcode!
- Furthermore, we have to change the instruction itself if we want to manipulate the address (for example for an array)
- We could save some space by, e.g. putting the address in a register, and accessing *relative* to that register. This is called **indirect** addressing. Here we have a base address stored in a register, and e.g. a constant offset in another register
- There are more advanced addressing modes, some of which we will see